Lectures
Display List

By
Tom Duff
Pixar Animation Studios
Emeryville, California
and
George Ledin Jr
Sonoma State University
Rohnert Park, California

What is Display List?

- What is it?
  - A group of OpenGL commands that have been stored for later execution.
  - Because a display list is a cache of commands, once it is created, it cannot be modified.
  - There is no facility to save the contents of a display list into a data file, and no facility to create a display list from a file. In this sense, a display list is designed for temporary use.
  - OpenGL allows you to create a display list that calls another list that hasn't been created yet. Nothing happens when the first list calls the second, undefined one.

Without display list, the standard rendering mode is immediate mode

- The standard rendering mode in OpenGL is known as immediate mode. Primitives are passed through the OpenGL pipeline as soon as they are defined in the program. They are then no longer in the system; only their image is on the screen. When something changes and the screen needs to be redrawn, we have to regenerate the primitives.

- This redisplay process can be very time-consuming. It can be especially slow when the application program (the client) is on one side of a network and the render and display process (the graphics server) is on the other.

Retained mode

- In retained mode graphics, collections of primitives and other information can be stored as objects on the server, thus avoid costly transfers and regeneration problems.

The advantage of display list

- The display list store their contents in an internal format that makes for fast display.

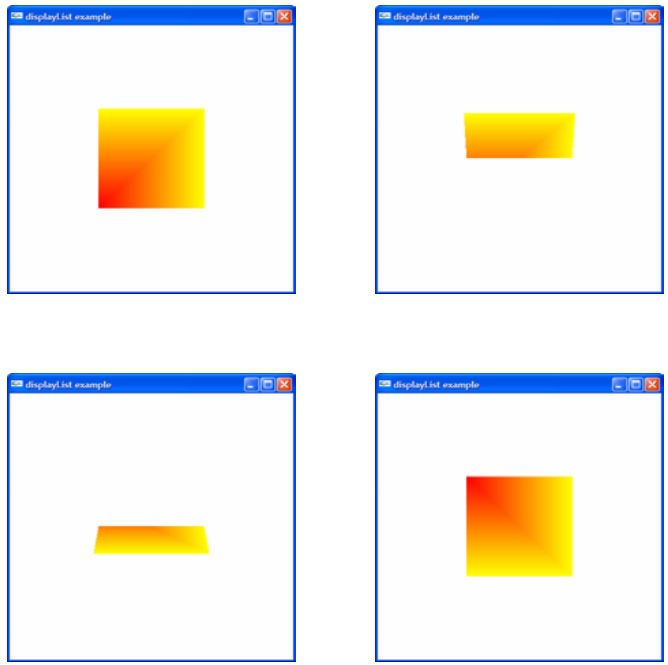
- The display list uses retained mode graphics, and improve the performance of OpenGL by storing data at the server side.

When to use display list?

- If you plan to redraw something multiple times.
- If you have a set of state changes to be applied multiple times.

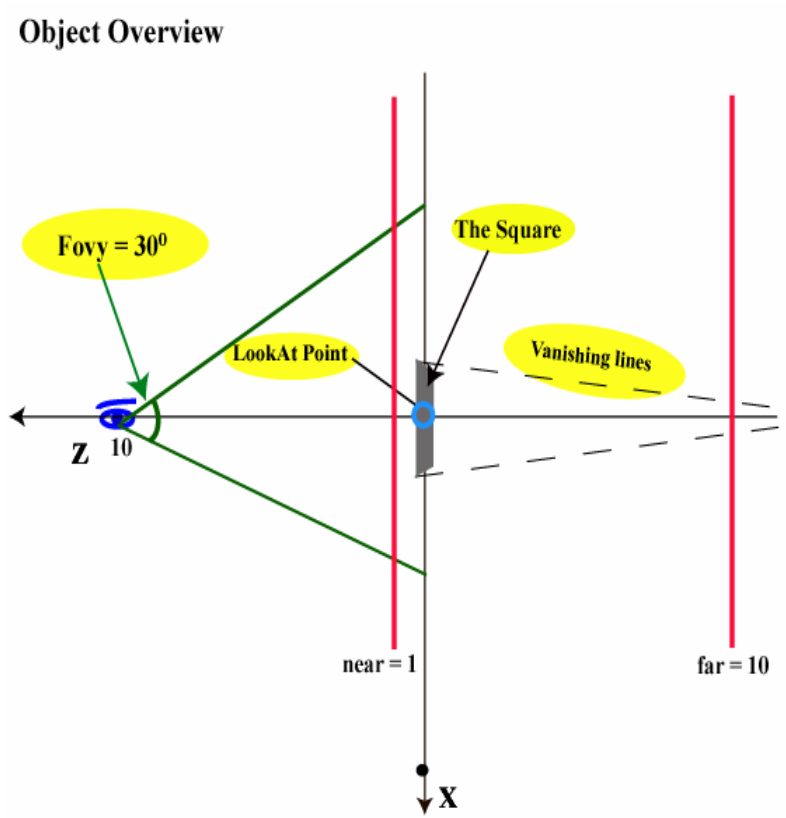
Case Study 1
Drawing a square and viewing it from different angles.

- How to draw a square and view it from different angles?
  - Store the square in a display list.
  - Whenever you change the model view matrix, execute the stored display list.



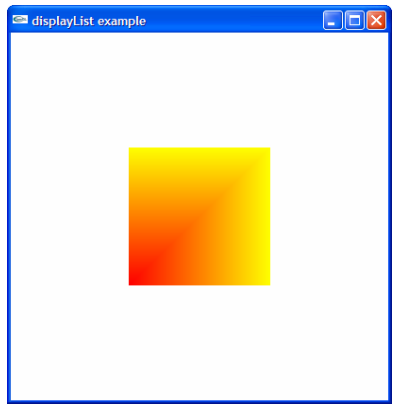
(1) Define a viewing perspective

```
glViewport(0, 0,
           (GLsizei) w, (GLsizei) h);

glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(30,
               (GLfloat)
               w/(GLfloat) h,
               1.0,
               100.0);

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(0, 0, 10, 0, 0, 0, 0, 1, 0);
```



(2) Define a square

```
// Define a square of two colors
static void squre(int numc, int numt)
{
     glBegin(GL_POLYGON);
           glColor3f(1,0,0);
            glVertex3f(-1, -1, 0);

            glColor3f(1,1,0);
            glVertex3f(-1,1,0);

            glVertex3f(1,1,0);
            glVertex3f(1,-1,0);
      glEnd();
}
```



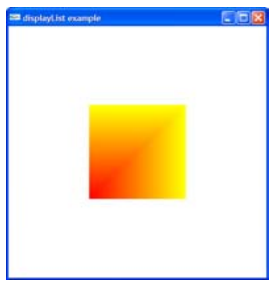
(3) Create a Display List

```
/* Create display list with Square and
    initialize state*/
GLuint theSquare;

static void init(void)
{
   theSquare = glGenLists (1);
   glNewList(theSquare, GL_COMPILE);
   square();
   glEndList();
    …
}
```

- **GLuint glGenLists(range)**
  - Specifies the number of continuous empty display lists to be generated
  - Returns display list name
- **Why do we need to use glGenLists to generate a display list index?**
  - Because we don't want to choose an index that is already created, and overwrite an existing display list.
  - Zero is returned if the requested number of indices isn't available, or if *range* is zero.

(3) Create Display List, continued…

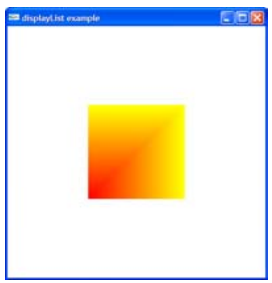
```
/* Create display list with Square
   and initialize state*/
GLuint theSquare;

static void init(void)
{
   theSquare = glGenLists (1);
   glNewList(theSquare,
   GL_COMPILE);
   squre();
   glEndList();
   …
}
```



- **void glNewList(listName, mode)**
  - list: display list name
  - Mode:
    - GL_COMPILE: compile command
    - GL_COMPILE_EXECUTE: commands are executed as they are compiled into display list
- **void glEndList()**
  - All subsequent commands are placed in the display list, in the order issued, until glEndList is called.

(4) Define Display CallBack function

```
void display(void)
{
   SetBackground(1,1,1,0); // white
   glCallList(theSquare);
   glFlush();
}

int main(int argc, char **argv)
{
   …
  glutDisplayFunc(display);
  …
}
```



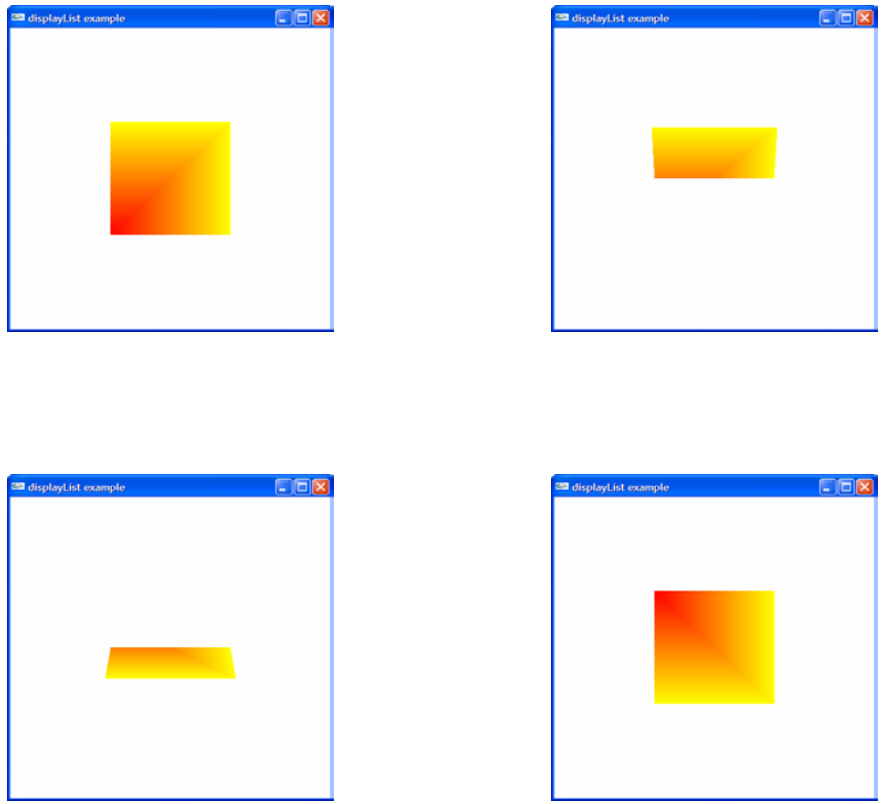
- void **glCallList** (GLuint *list*);
  - list: The integer name of the display list to be executed
  - glCallList causes the named display list to be executed.

(4) Define when to invoke the display – keyboard input in this case

```
void keyboard(unsigned char key, int x, int
     y)
{
   switch (key) {
   case 'x': case 'X':
      glRotatef(30,1,0,0.0);
      glutPostRedisplay();
      break;
   case 'y': case 'Y':
      glRotatef(30,0,1,0);
      glutPostRedisplay();
      break;
   case 'z': case 'Z':
     glRotatef(30,0,0,1);
     glutPostRedisplay();
     break;
   case 'i': case 'I':
      glLoadIdentity();
      gluLookAt(0, 0, 10, 0, 0, 0, 0, 1, 0);
      glutPostRedisplay();
      break;
   case 27:
      exit(0);
      break;
   }
}
```

- void glutPostRedisplay ();
  - glutPostRedisplay marks the normal plane of current window as needing to be redisplayed.
  - After either call, the next iteration through glutMainLoop, the window's display callback will be called to redisplay the window's normal plane.
- What does the code do?
  - If user hits “x” or “X” key, the square will be rotated around the X axis.
  - If user hits “y” or “Y” key, the square will be rotated around the Y axis.
  - If user hits “z” or “Z” key, the square will be rotated around the Z axis.
  - If user hits “i” or “I” key, the square will be set to its original state.

Output when user clicks “X”
Note: the square rotates around the X axis



Case Study 2
Use display list to draw many triangles

- How to draw a triangle multiple times?

  - Create a display list contains OpenGL commands to draw a triangle. (init())
  - In the display call back routine (display()), executes the display list as many times as is needed.

(1) Define a red triangle

```
GLuint listName;
static void init (void)
    {
    listName = glGenLists (1);
    glNewList (listName, GL_COMPILE);

    glColor3f (1.0, 0.0, 0.0); //red

    glShadeModel (GL_FLAT);
    /* current color red */
    glBegin (GL_TRIANGLES);
          glVertex2f (0.0, 0.0);
          glVertex2f (1.0, 0.0);
          glVertex2f (0.0, 1.0);
    glEnd ();

    glTranslatef (1.5, 0.0, 0.0); /* move
    position */
    glEndList ();
    }
```

- **Observations:**
  - The triangle is drawn in red with flat shading.
  - glTranslatef() routine in the display list alters the position of the next object to be drawn

(2) Call Display List in display call back

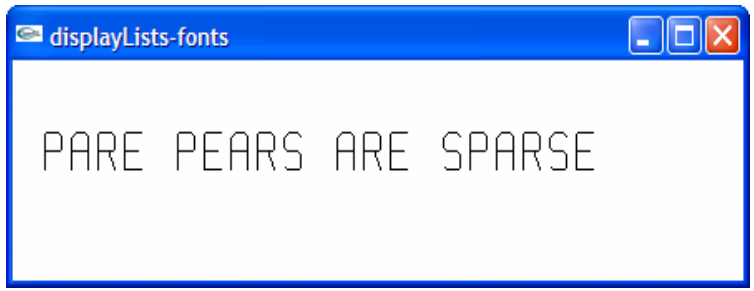
```
void display(void)
     { GLuint i;
     glClear(GL_COLOR_BUFFER_BIT);
     glColor3f (0.0, 1.0, 0.0); /* current color green */
     for (i = 0; i < 10; i++) /* draw 10 triangles */
        glCallList (listName);

     glFlush ();
     }
```

Case Study 3
Executing Multiple Display Lists to create stroked fonts

- Display Lists are very useful to create fonts.

- Goal of study case:
  - In this case study, we will create a display list of characters “A”, “E”, “P”, “R”, “S”, and “ “ (blank space).

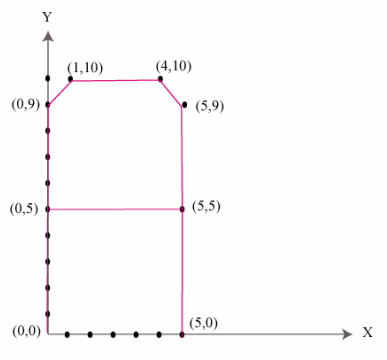
  - Use this display list to write a sentence “A SPARE APPEARS AS APES PREPARE RARE REPPERS”.



Create stroked fonts
(1) create fonts – example of creating “A”

```
#define PT          1 // point
#define STROKE      2
#define END         3

typedef struct charpoint
{ GLfloat x, y;
int type; } CP;
CP Adata[] = {
    { 0, 0, PT}, {0, 9, PT}, {1, 10,
    PT},
    {4, 10, PT}, {5, 9, PT}, {5, 0,
    STROKE},
    {0, 5, PT}, {5, 5, END}
};
```

- How to create font ‘A’:




```
/*  drawLetter() interprets the instructions
     from the array
 *   for that letter and renders the letter with
     line segments.
 */
static void drawLetter(CP *l)
{
   glBegin(GL_LINE_STRIP);
   while (1) {
      switch (l->type) {
         case PT:
            glVertex2fv(&l->x);
            break;
         case STROKE:
            glVertex2fv(&l->x);
            glEnd();
            glBegin(GL_LINE_STRIP);
            break;
         case END:
            glVertex2fv(&l->x);
            glEnd();
            glTranslatef(8.0, 0.0, 0.0);
            return;
      }
      l++;
   }
}
```

Create stroked fonts
(2)create display lists for fonts

```
/* Create a display list for each of 6
    characters */
void initStrokedFont(void){
    GLuint base;
    base = glGenLists(128);
    glListBase(base);
    glNewList(base+'A', GL_COMPILE);
          drawLetter(Adata);
    glEndList();
    glNewList(base+'E', GL_COMPILE);
          drawLetter(Edata);
    glEndList();
    glNewList(base+'P', GL_COMPILE);
          drawLetter(Pdata);
    glEndList();
    glNewList(base+'R', GL_COMPILE);
          drawLetter(Rdata);
    glEndList();
    glNewList(base+'S', GL_COMPILE);
          drawLetter(Sdata);
    glEndList();
    glNewList(base+' ', GL_COMPILE); /*
    space character */
          glTranslatef(8.0, 0.0, 0.0);
    glEndList();
    }
```

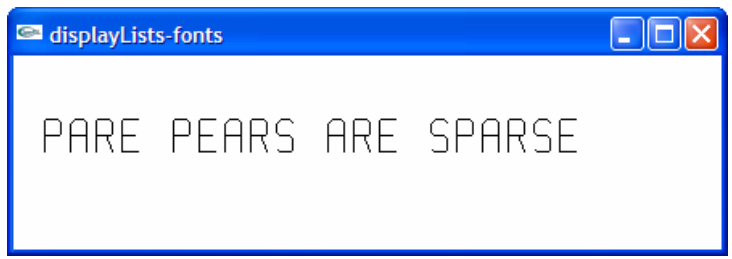
- The glGenLists() command allocates 128 contiguous display-list indices.
- The first of the contiguous indices becomes the display-list base.
  - You can specify this initial index by using glListBase() before calling glCallLists().
- Each display-list index is the sum of the base and the ASCII value of that letter.
- In this example, only “A”, “E”, “R”, “S” and “ “ (space) characters are created.

Create stroked fonts
- (3) Call glCallLists() to execute display lists.

```
/* Each ascii code is the offsets to the display lists. */

void printStrokedString(GLbyte *s)
    {
    GLint len = strlen(s);
    glCallLists(len, GL_BYTE, s);
    }
```

```
char *test1 = "RARE PEARS ARE SPARSE";

void display(void)
{
…
printStrokedString(test1);
..
}
```

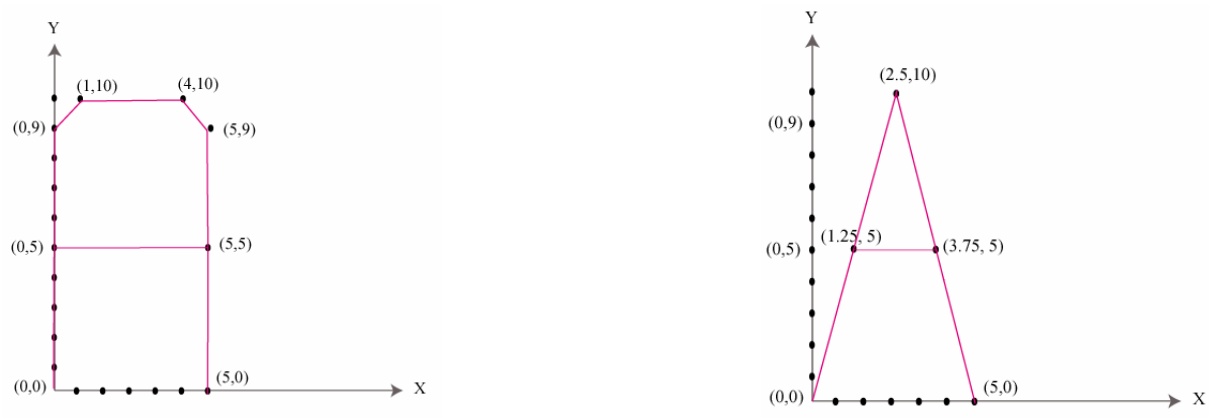
- void **glCallLists**(GLsizei *n*, GLenum *type*, const GLvoid **lists*)
- **PARAMETERS**
  - *n*: Specifies the number of display lists to be executed.
  - *type*: Specifies the type of values in *lists*. Symbolic constants **GL_BYTE**, **GL_UNSIGNED_BYTE**, and etc are accepted.
  - *lists*: Specifies the address of an array of name offsets in the display list. The pointer type is void because the offsets can be bytes, shorts, ints, or floats, depending on the value of *type*.



How can there be several fonts?

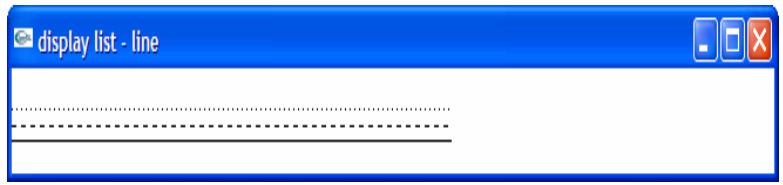
- To have several such fonts, you would need to establish a different initial display-list index for each font.

Example of two different fonts of "A"



Case Study 4
Using a display list for drawing different styles of lines

- Goal:
  - Create three lines.
  - Each line is drawn in a different style.



Case Study 4
(1) Creating display list for three different line styles

```
/* define display list for lines*/
GLuint offset;
offset = glGenLists(3);

glNewList (offset, GL_COMPILE);
    glDisable (GL_LINE_STIPPLE);
glEndList ();

glNewList (offset+1, GL_COMPILE);
    glEnable (GL_LINE_STIPPLE);
    glLineStipple (1, 0x0F0F);
glEndList ();

glNewList (offset+2, GL_COMPILE);
    glEnable (GL_LINE_STIPPLE);
    glLineStipple (1, 0x1111);
glEndList ();
```

Case Study 4
(2) Calling a display list to draw three lines of different styles

```
/* use display list to draw different lines */

#define drawOneLine(x1,y1,x2,y2) glBegin(GL_LINES);
    glVertex2f ((x1),(y1)); glVertex2f ((x2),(y2)); glEnd();

void display(void){
   GLuint i;

   glClear (GL_COLOR_BUFFER_BIT);
   glColor3f (0.0, 0.0, 0.0);  /*  black  */

   glCallList (offset);
   drawOneLine(0,0.1,15,0.1);
   glCallList (offset+1);
   drawOneLine(0,0.4,15,0.4);
   glCallList (offset+2);
   drawOneLine(0,0.7,15,0.7);
   glFlush ();
}
```

Managing Display List Indices

- So far, we've recommended the use of glGenLists() to obtain unused display-list indices.

- If you insist upon avoiding glGenLists(), then be sure to use glIsList() to determine whether a specific index is in use.

  - GLboolean glIsList(GLuint *list*);
    - Returns GL_TRUE if *list* is already used for a display list and GL_FALSE otherwise.

How to delete a display list?

- You can explicitly delete a specific display list or a contiguous range of lists with glDeleteLists(). Using glDeleteLists() makes those indices available again.
  - void glDeleteLists(GLuint *list*, GLsizei *range*);
    - Deletes a *range* of display lists, starting at the index specified by *list*.
    - An attempt to delete a list that has never been created is ignored.

Case Study 5
What is the color of vertex? – Black or Red?

```
GLfloat color_vector[3]
          = {0.0, 0.0, 0.0}; // black

glNewList(1, GL_COMPILE);
    glColor3fv(color_vector);
glEndList();

color_vector[0] = 1.0;          // red
```

- Answer : Black
- Reason:
  - The subsequent change of the value of the *color_vector* array to red (1.0, 0.0, 0.0) has no effect on the display list.
  - Because the display list contains the values that were in effect when it was created.

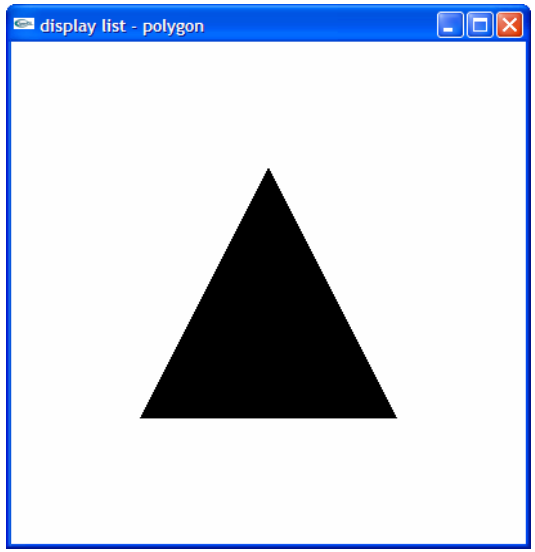
Hierarchical Display Lists

- You can create a hierarchical display list, which is a dissplay list that executes another display list by calling glCallList() between a glNewList and glEndList().

- OpenGL allows you to create a display list that calls another list that hasn't been created yet.
  - Nothing happens when the first list calls the second undefined one.

Case Study 6
Using a Display List for easily changing geometric parameters

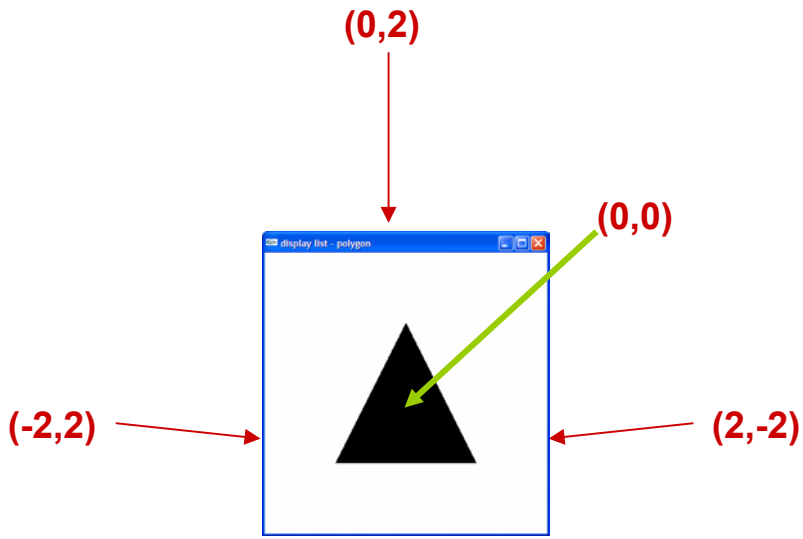
- Goal:
  - Use a hierarchical display list to create a three vertex polygon (a triangle in this case)



Case Study 6
(1) Create a polygon display list using a hierarchical display list

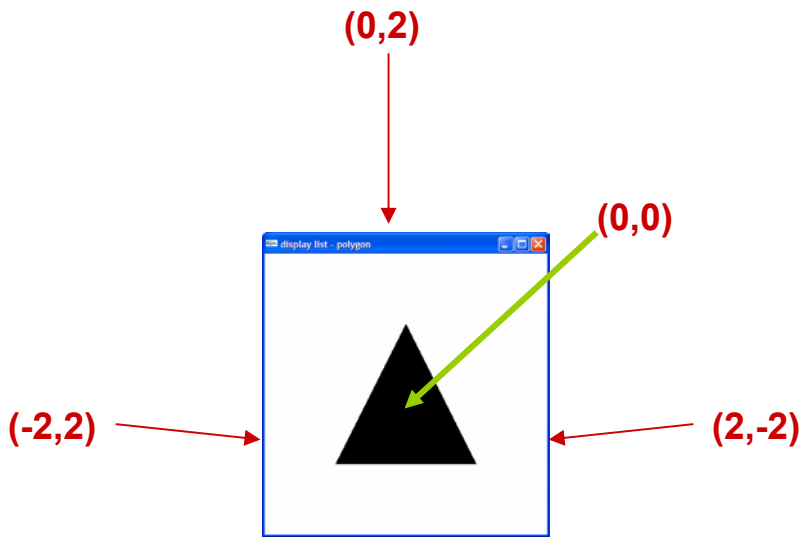
```
/* create display lists for the 3
    vertices */
glNewList(1,GL_COMPILE);
    glVertex3f(-2,-2);
glEndList();

glNewList(2,GL_COMPILE);
    glVertex3f(2,-2);
glEndList();

glNewList(3,GL_COMPILE);
    glVertex3f(0,2);
glEndList();

/* create a display list of polygon using
   previous vertex display lists*/

glNewList(4,GL_COMPILE);
    glBegin(GL_POLYGON);
          glCallList(1);
          glCallList(2);
          glCallList(3);
          glEnd();
glEndList();
```

- Note:
  - -To put a polygon in a display list while allowing yourself to be able to easily edit its vertices.
  - -To render the polygon, call display list number 4.



Case Study 6
(2) Call display list to create the polygon

```
void display(void)
{
   GLuint i;

   glClear (GL_COLOR_BUFFER_BIT);

   /* black color for the triangle*/
   glColor3f (0.0, 0.0, 0.0);

   /* call polygon display list */
   glCallList(4);

   glFlush ();
}
```



Advantage of using hierachical display lists to create geometric objects

- To edit a vertex, you need only recreate the single display list corresponding to that vertex.

- Since an index number uniquely identifies a display list, creating one with the same index as an existing one automatically deletes the old one.

Do hierarchical display lists improve memory usage or performance?

- Not necessarily.
- Keep in mind that this technique doesn't necessarily provide optimal memory usage or peak performance, but it's acceptable and useful in some cases.
- However, hierarchical display list is useful for an object made of different components, especially when some of those components are used multiple times.

Limitations on the nesting level of display lists.

- To avoid infinite recursion, there's a limit on the nesting level of display lists.
- The limit is at least 64, but it might be higher, depending on the implementation .
- To determine the nesting limit for your implementation of OpenGL, call
  - glGetIntegerv(GL_MAX_LIST_NESTING, GLint *data);

Display list disadvantages

- The immutability of the contents of a display list.
- The execution of display lists isn't slower than executing the commands contained within them individually.
  - Very small lists may not perform well since there is some overhead when executing a list.

Display List cannot store all OpenGL commands

- **The following commands can not be stored in DisplayList:**

```
glColorPointer()
glFlush()
glNormalPointer()
glDeleteLists()
glGenLists()
glPixelStore()
glDisableClientState()
glGet*()
glReadPixels()
glEdgeFlagPointer()
glIndexPointer()
glRenderMode()
glEnableClientState()
glInterleavedArrays()
glSelectBuffer()
glFeedbackBuffer()
glIsEnabled()
glTexCoordPointer()
glFinish()
glIsList()
glVertexPointer()
```

- **Why?**
  - When using OpenGL across a network, the client on one machine, the server on another.
  - The display list resides on the server. The server can't rely on the client for any information.
  - Querying commands such as glGet*() and glIs*() will return data which server will not know where to store them.

Display List cannot store all OpenGL commands, continued …

- **The following commands can not be stored in DisplayList:**

```
glColorPointer()
glFlush()
glNormalPointer()
glDeleteLists()
glGenLists()
glPixelStore()
glDisableClientState()
glGet*()
glReadPixels()
glEdgeFlagPointer()
glIndexPointer()
glRenderMode()
glEnableClientState()
glInterleavedArrays()
glSelectBuffer()
glFeedbackBuffer()
glIsEnabled()
glTexCoordPointer()
glFinish()
glIsList()
glVertexPointer()
```

- **Why?**
  - Commands that change client's state, such as glPixelStore(), glSelectBuffer() can't be stored in display list.

Display List cannot store all OpenGL commands, continued …

- **The following commands can not be stored in DisplayList:**

  glColorPointer()
  glFlush()
  glNormalPointer()
  glDeleteLists()
  glGenLists()
  glPixelStore()
  glDisableClientState()
  glGet*()
  glReadPixels()
  glEdgeFlagPointer()
  glIndexPointer()
  glRenderMode()
  glEnableClientState()
  glInterleavedArrays()
  glSelectBuffer()
  glFeedbackBuffer()
  glIsEnabled()
  glTexCoordPointer()
  glFinish()
  glIsList()
  glVertexPointer()

- **Why?**
  - Commands that depend on client state, such as glVertexPointer(), glColorPointer(), and glInterleavedArrays() can not be stored in display list.

Display List cannot store all OpenGL commands

- **The following commands can not be stored in DisplayList:**

```
glColorPointer()
glFlush()
glNormalPointer()
glDeleteLists()
glGenLists()
glPixelStore()
glDisableClientState()
glGet*()
glReadPixels()
glEdgeFlagPointer()
glIndexPointer()
glRenderMode()
glEnableClientState()
glInterleavedArrays()
glSelectBuffer()
glFeedbackBuffer()
glIsEnabled()
glTexCoordPointer()
glFinish()
glIsList()
glVertexPointer()
```

- **Why?**
  - Routines that rely upon client state — such as glFlush() and glFinish()— can't be stored in a display list because they depend upon the client state that is in effect when they are executed.